Contents

Egress using Wildcard Hosts

12 minute read   ✓ page test

The Accessing External Services task and the Configure an Egress Gateway example describe how to configure egress traffic for specific hostnames, like `edition.cnn.com`. This example shows how to enable egress traffic for a set of hosts in a common domain, for example `*.wikipedia.org`, instead of configuring each and every host separately.

Background

Suppose you want to enable egress traffic in Istio for the `wikipedia.org` sites in all languages. Each version of `wikipedia.org` in a particular language has its own hostname, e.g., `en.wikipedia.org` and `de.wikipedia.org` in the English and the German languages, respectively. You want to enable egress traffic by common configuration items for all the Wikipedia sites, without the need to specify every language's site separately.

Before you begin

- Install Istio using the `demo` configuration profile and with the blocking-by-default outbound traffic policy:

  ```
  $ istioctl install --set profile=demo --set meshConfig.outboundTraffic
  Policy.mode=REGISTRY_ONLY
  ```

  You can run this task on an Istio configuration other than the `demo` profile as long as you make sure to deploy the Istio egress gateway, enable Envoy's access logging, and apply the blocking-by-default outbound traffic policy in your installation. You will also need to add the second gateway using your own `IstioOperator` CR instead of the one shown in setup egress gateway with SNI proxy.

- Deploy the sleep sample app to use as a test source for sending requests. If you have automatic sidecar injection enabled, run the following command to deploy the sample app:

```
$ kubectl apply -f @samples/sleep/sleep.yaml@
```

Otherwise, manually inject the sidecar before deploying the `sleep` application with the following command:

```
$ kubectl apply -f <(istioctl kube-inject -f @samples/sleep/sleep.yaml
@)
```

You can use any pod with `curl` installed as a test source.

- Set the `SOURCE_POD` environment variable to the name of your source pod:

```
$ export SOURCE_POD=$(kubectl get pod -l app=sleep -o jsonpath={.items
..metadata.name})
```

Configure direct traffic to a wildcard host

The first, and simplest, way to access a set of hosts within a common domain is by configuring a simple `ServiceEntry` with a wildcard host and calling the services directly from the sidecar. When calling services directly (i.e., not via an egress gateway), the configuration for a wildcard host is no different than that of any other (e.g., fully qualified) host, only much more convenient when there are many hosts within the common domain.

Note that the configuration below can be easily bypassed by a malicious application. For a secure egress traffic control, direct the traffic through an egress gateway.

Note that the `DNS` resolution cannot be used for wildcard hosts. This is why the `NONE` resolution (omitted since it is the default) is used in the service entry below.

1. Define a `ServiceEntry` for `*.wikipedia.org`:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: wikipedia
spec:
  hosts:
  - "*.wikipedia.org"
  ports:
  - number: 443
    name: https
    protocol: HTTPS
EOF
```

2. Send HTTPS requests to https://en.wikipedia.org and https://de.wikipedia.org:

```
$ kubectl exec "$SOURCE_POD" -c sleep -- sh -c 'curl -s https://en.wik
ipedia.org/wiki/Main_Page | grep -o "<title>.*</title>"; curl -s https
://de.wikipedia.org/wiki/Wikipedia:Hauptseite | grep -o "<title>.*</ti
tle>"'
<title>Wikipedia, the free encyclopedia</title>
<title>Wikipedia – Die freie Enzyklopädie</title>
```

Cleanup direct traffic to a wildcard host

```
$ kubectl delete serviceentry wikipedia
```

Configure egress gateway traffic to a wildcard host

The configuration for accessing a wildcard host via an egress gateway depends on whether or not the set of wildcard domains are served by a single common host. This is the case for **.wikipedia.org*. All of the language-specific sites are served by every one of the *wikipedia.org* servers. You can route the traffic to an IP of any **.wikipedia.org* site, including *www.wikipedia.org*, and it will manage to serve any specific site.

In the general case, where all the domain names of a wildcard are not served by a single hosting server, a more complex configuration is required.

Wildcard configuration for a single hosting server

When all wildcard hosts are served by a single server, the configuration for egress gateway-based access to a wildcard host is very similar to that of any host, with one exception: the configured route destination will not be the same as the configured host, i.e., the wildcard. It will instead be configured with the host of the single server for the set of domains.

1. Create an egress `Gateway` for **.wikipedia.org*, a destination rule and a virtual service to direct the traffic through the egress gateway and from the egress gateway to the external service.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: istio-egressgateway
spec:
  selector:
    istio: egressgateway
  servers:
  - port:
      number: 443
      name: https
      protocol: HTTPS
    hosts:
    - "*.wikipedia.org"
    tls:
      mode: PASSTHROUGH
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: egressgateway-for-wikipedia
```

```
spec:
  host: istio-egressgateway.istio-system.svc.cluster.local
  subsets:
    - name: wikipedia
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: direct-wikipedia-through-egress-gateway
spec:
  hosts:
  - "*.wikipedia.org"
  gateways:
  - mesh
  - istio-egressgateway
  tls:
  - match:
    - gateways:
      - mesh
      port: 443
      sniHosts:
      - "*.wikipedia.org"
    route:
    - destination:
        host: istio-egressgateway.istio-system.svc.cluster.local
        subset: wikipedia
        port:
          number: 443
      weight: 100
  - match:
    - gateways:
      - istio-egressgateway
      port: 443
      sniHosts:
      - "*.wikipedia.org"
    route:
    - destination:
        host: www.wikipedia.org
        port:
          number: 443
      weight: 100
EOF
```

2. Create a `ServiceEntry` for the destination server, *www.wikipedia.org*.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: www-wikipedia
spec:
  hosts:
  - www.wikipedia.org
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  resolution: DNS
EOF
```

3. Send HTTPS requests to https://en.wikipedia.org and https://de.wikipedia.org:

```
$ kubectl exec "$SOURCE_POD" -c sleep -- sh -c 'curl -s https://en.wik
ipedia.org/wiki/Main_Page | grep -o "<title>.*</title>"; curl -s https
://de.wikipedia.org/wiki/Wikipedia:Hauptseite | grep -o "<title>.*</ti
tle>"'
<title>Wikipedia, the free encyclopedia</title>
<title>Wikipedia – Die freie Enzyklopädie</title>
```

4. Check the statistics of the egress gateway’s proxy for the counter that corresponds to your requests to **.wikipedia.org*. If Istio is deployed in the `istio-system` namespace, the command to print the counter is:

```
$ kubectl exec "$(kubectl get pod -l istio=egressgateway -n istio-syst
em -o jsonpath='{.items[0].metadata.name}')" -c istio-proxy -n istio-s
ystem -- pilot-agent request GET clusters | grep '^outbound|443||www.w
ikipedia.org.*cx_total:'
outbound|443||www.wikipedia.org::208.80.154.224:443::cx_total::2
```

Cleanup wildcard configuration for a single hosting server

```
$ kubectl delete serviceentry www-wikipedia
$ kubectl delete gateway istio-egressgateway
$ kubectl delete virtualservice direct-wikipedia-through-egress-gateway
$ kubectl delete destinationrule egressgateway-for-wikipedia
```

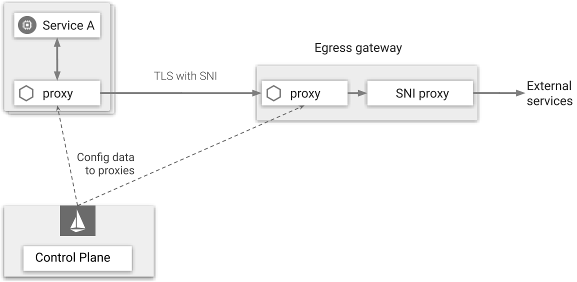
Wildcard configuration for arbitrary domains

The configuration in the previous section worked because all the **.wikipedia.org* sites can be served by any one of the *wikipedia.org* servers. However, this is not always the case. For example, you may want to configure egress control for access to more general wildcard domains like `*.com` or `*.org`.

Configuring traffic to arbitrary wildcard domains introduces a challenge for Istio gateways. In the previous section you directed the traffic to *www.wikipedia.org*, which was made known to your gateway during configuration. The gateway, however, would not know the IP address of any arbitrary host it receives in a request. This is due to a limitation of Envoy, the proxy used by the default Istio egress gateway. Envoy routes traffic either to predefined hosts, predefined IP addresses, or to the original destination IP address of the request. In the gateway case, the original destination IP of the request is lost since the request is first routed to the egress gateway and its destination IP address is the IP address of the gateway.

Consequently, the Istio gateway based on Envoy cannot route traffic to an arbitrary host that is not preconfigured, and therefore is unable to perform traffic control for arbitrary wildcard domains. To enable such traffic control for HTTPS, and for any TLS, you need to deploy an SNI forward proxy in addition to Envoy. Envoy will route the requests destined for a wildcard domain to the SNI forward proxy, which, in turn, will forward the requests to the destination specified by the SNI value.

The egress gateway with SNI proxy and the related parts of the Istio architecture are shown in the following diagram:



Egress Gateway with SNI proxy

The following sections show you how to redeploy the egress gateway with an SNI proxy and then configure Istio to route HTTPS traffic through the gateway to arbitrary wildcard domains.

Setup egress gateway with SNI proxy

In this section you deploy an egress gateway with an SNI proxy in addition to the standard Istio Envoy proxy. This example uses Nginx for the SNI proxy, although any SNI proxy that is capable of routing traffic according to arbitrary, not-preconfigured, SNI values would do. The SNI proxy will listen on port `8443`, although you can use any port other than the ports specified for the egress `Gateway` and for the `VirtualServices` bound to it. The SNI proxy will forward the traffic to port `443`.

1. Create a configuration file for the Nginx SNI proxy. You may want to edit the file to specify additional Nginx settings, if required. Note that the `listen` directive of the `server` specifies port `8443`, its `proxy_pass` directive uses `ssl_preread_server_name` with port `443` and `ssl_preread` is `on` to enable `SNI` reading.

   ```
   $ cat <<EOF > ./sni-proxy.conf
   # setup custom path that do not require root access
   pid /tmp/nginx.pid;

   events {
   }

   stream {
     log_format log_stream '\$remote_addr [\$time_local] \$protocol [\$ss
   l_preread_server_name]'
     '\$status \$bytes_sent \$bytes_received \$session_time';
   ```

```
  access_log /var/log/nginx/access.log log_stream;
  error_log  /var/log/nginx/error.log;

  # tcp forward proxy by SNI
  server {
    resolver 8.8.8.8 ipv6=off;
    listen       127.0.0.1:18443;
    proxy_pass   \$ssl_preread_server_name:443;
    ssl_preread  on;
  }
}
EOF
```

2. Create a Kubernetes `ConfigMap` to hold the configuration of the Nginx SNI proxy:

   ```
   $ kubectl create configmap egress-sni-proxy-configmap -n istio-system
   --from-file=nginx.conf=./sni-proxy.conf
   ```

3. Create an `IstioOperator` CR to add a new egress gateway

with SNI proxy:

```
$ istioctl manifest generate -f - <<EOF > ./egressgateway-with-sni-pro
xy.yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  # Only generate a gateway component defined below.
  # Using this with "istioctl install" will reconcile and remove exist
ing control-plane components.
  # Instead use "istioctl manifest generate" or "kubectl create" if us
ing the istio operator.
  profile: empty
  components:
    egressGateways:
    - name: istio-egressgateway-with-sni-proxy
      enabled: true
      label:
        app: istio-egressgateway-with-sni-proxy
        istio: egressgateway-with-sni-proxy
      k8s:
        service:
          ports:
          - port: 443
            targetPort: 8443
            name: https
        overlays:
        - kind: Deployment
          name: istio-egressgateway-with-sni-proxy
          patches:
          - path: spec.template.spec.containers[-1]
            value: |
              name: sni-proxy
              image: nginx
              volumeMounts:
              - name: sni-proxy-config
                mountPath: /etc/nginx
                readOnly: true
              securityContext:
                runAsNonRoot: true
                runAsUser: 101
          - path: spec.template.spec.volumes[-1]
            value: |
              name: sni-proxy-config
              configMap:
                name: egress-sni-proxy-configmap
                defaultMode: 292 # 0444
EOF
```

4. Deploy the new gateway:

```
$ kubectl apply -f ./egressgateway-with-sni-proxy.yaml
```

5. Verify that the new egress gateway is running. Note that the pod has two containers (one is the Envoy proxy and the second one is the SNI proxy).

```
$ kubectl get pod -l istio=egressgateway-with-sni-proxy -n istio-syste
m
NAME                                                  READY     STATUS
    RESTARTS   AGE
istio-egressgateway-with-sni-proxy-79f6744569-pf9t2   2/2       Runnin
g   0          17s
```

6. Create a service entry with a static address equal to 127.0.0.1 (`localhost`), and disable mutual TLS for traffic directed to the new service entry:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: sni-proxy
spec:
  hosts:
  - sni-proxy.local
  location: MESH_EXTERNAL
  ports:
  - number: 18443
    name: tcp
    protocol: TCP
  resolution: STATIC
  endpoints:
  - address: 127.0.0.1
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: disable-mtls-for-sni-proxy
spec:
  host: sni-proxy.local
  trafficPolicy:
    tls:
      mode: DISABLE
EOF
```

Configure traffic through egress

gateway with SNI proxy

1. Define a `ServiceEntry` for `*.wikipedia.org`:

    ```
    $ cat <<EOF | kubectl create -f -
    apiVersion: networking.istio.io/v1alpha3
    kind: ServiceEntry
    metadata:
      name: wikipedia
    spec:
      hosts:
      - "*.wikipedia.org"
      ports:
      - number: 443
        name: tls
        protocol: TLS
    EOF
    ```

2. Create an egress `Gateway` for **.wikipedia.org*, port 443,

protocol TLS, and a virtual service to direct the traffic destined for **.wikipedia.org* through the gateway.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: istio-egressgateway-with-sni-proxy
spec:
  selector:
    istio: egressgateway-with-sni-proxy
  servers:
  - port:
      number: 443
      name: tls-egress
      protocol: TLS
    hosts:
    - "*.wikipedia.org"
    tls:
      mode: ISTIO_MUTUAL
---
```

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: egressgateway-for-wikipedia
spec:
  host: istio-egressgateway-with-sni-proxy.istio-system.svc.cluster.lo
cal
  subsets:
    - name: wikipedia
      trafficPolicy:
        loadBalancer:
          simple: ROUND_ROBIN
        portLevelSettings:
        - port:
            number: 443
          tls:
            mode: ISTIO_MUTUAL
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: direct-wikipedia-through-egress-gateway
```

```
spec:
  hosts:
  - "*.wikipedia.org"
  gateways:
  - mesh
  - istio-egressgateway-with-sni-proxy
  tls:
  - match:
    - gateways:
      - mesh
      port: 443
      sniHosts:
      - "*.wikipedia.org"
    route:
    - destination:
        host: istio-egressgateway-with-sni-proxy.istio-system.svc.clus
ter.local
        subset: wikipedia
        port:
          number: 443
      weight: 100
  tcp:
```

```
  - match:
    - gateways:
      - istio-egressgateway-with-sni-proxy
      port: 443
    route:
    - destination:
        host: sni-proxy.local
        port:
          number: 18443
      weight: 100
---
# The following filter is used to forward the original SNI (sent by th
e application) as the SNI of the
# mutual TLS connection.
# The forwarded SNI will be will be used to enforce policies based on
the original SNI value.
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: forward-downstream-sni
spec:
  configPatches:
```

```
  - applyTo: NETWORK_FILTER
    match:
      context: SIDECAR_OUTBOUND
      listener:
        portNumber: 443
        filterChain:
          filter:
            name: istio.stats
    patch:
      operation: INSERT_BEFORE
      value:
         name: forward_downstream_sni
         config: {}
EOF
```

3. Add an `EnvoyFilter` to the gateway, to prevent it from being deceived.

```
$ kubectl apply -n istio-system -f - <<EOF
# The following filter verifies that the SNI of the mutual TLS connect
ion is
```

```
# identical to the original SNI issued by the client (the SNI used for
 routing by the SNI proxy).
# The filter prevents the gateway from being deceived by a malicious c
lient: routing to one SNI while
# reporting some other value of SNI. If the original SNI does not matc
h the SNI of the mutual TLS connection,
# the filter will block the connection to the external service.
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: egress-gateway-sni-verifier
spec:
  workloadSelector:
    labels:
      app: istio-egressgateway-with-sni-proxy
  configPatches:
  - applyTo: NETWORK_FILTER
    match:
      context: GATEWAY
      listener:
        portNumber: 443
        filterChain:
          filter:
            name: istio.stats
    patch:
      operation: INSERT_BEFORE
      value:
         name: sni_verifier
         config: {}
EOF
```

4. Send HTTPS requests to https://en.wikipedia.org and https://de.wikipedia.org:

   ```
   $ kubectl exec "$SOURCE_POD" -c sleep -- sh -c 'curl -s https://en.wik
   ipedia.org/wiki/Main_Page | grep -o "<title>.*</title>"; curl -s https
   ://de.wikipedia.org/wiki/Wikipedia:Hauptseite | grep -o "<title>.*</ti
   tle>"'
   <title>Wikipedia, the free encyclopedia</title>
   <title>Wikipedia – Die freie Enzyklopädie</title>
   ```

5. Check the log of the egress gateway's Envoy proxy. If Istio

is deployed in the `istio-system` namespace, the command to print the log is:

```
$ kubectl logs -l istio=egressgateway-with-sni-proxy -c istio-proxy -n
 istio-system
```

You should see lines similar to the following:

```
[2019-01-02T16:34:23.312Z] "- - -" 0 - 578 79141 624 - "-" "-" "-" "-"
 "127.0.0.1:18443" outbound|18443||sni-proxy.local 127.0.0.1:55018 172
.30.109.84:443 172.30.109.112:45346 en.wikipedia.org
[2019-01-02T16:34:24.079Z] "- - -" 0 - 586 65770 638 - "-" "-" "-" "-"
 "127.0.0.1:18443" outbound|18443||sni-proxy.local 127.0.0.1:55034 172
.30.109.84:443 172.30.109.112:45362 de.wikipedia.org
```

6. Check the logs of the SNI proxy. If Istio is deployed in the `istio-system` namespace, the command to print the log is:

```
$ kubectl logs -l istio=egressgateway-with-sni-proxy -n istio-system -
c sni-proxy
127.0.0.1 [01/Aug/2018:15:32:02 +0000] TCP [en.wikipedia.org]200 81513
 280 0.600
127.0.0.1 [01/Aug/2018:15:32:03 +0000] TCP [de.wikipedia.org]200 67745
 291 0.659
```

Cleanup wildcard configuration for arbitrary domains

1. Delete the configuration items for **.wikipedia.org*:

```
$ kubectl delete serviceentry wikipedia
$ kubectl delete gateway istio-egressgateway-with-sni-proxy
$ kubectl delete virtualservice direct-wikipedia-through-egress-gatewa
y
$ kubectl delete destinationrule egressgateway-for-wikipedia
$ kubectl delete --ignore-not-found=true envoyfilter forward-downstrea
m-sni
$ kubectl delete --ignore-not-found=true envoyfilter -n istio-system e
gress-gateway-sni-verifier
```

2. Delete the configuration items for the `egressgateway-with-sni-proxy` deployment:

```
$ kubectl delete serviceentry sni-proxy
$ kubectl delete destinationrule disable-mtls-for-sni-proxy
$ kubectl delete configmap egress-sni-proxy-configmap -n istio-system
$ kubectl delete -f ./egressgateway-with-sni-proxy.yaml
```

3. Remove the configuration files you created:

```
$ rm ./sni-proxy.conf ./egressgateway-with-sni-proxy.yaml
```

Cleanup

- Shutdown the `sleep` service:

  ```
  $ kubectl delete -f @samples/sleep/sleep.yaml@
  ```

- Uninstall Istio from your cluster:

  ```
  $ istioctl x uninstall --purge
  ```